

Logic Programming Engineering – Assignment 11

Dr.-Ing. Sascha Klüppelholz

Exercise 11.1 [Operator specification]

- a) Define a predicate `operator/1` that allows for enumerating the predefined operators. The definition should rely on `current_op/3`.

Example:

```
?- operator(Op).  
Op = ($) ;  
Op = (volatile) ;  
Op = (@=) ;  
⋮
```

- b) Use `current_op/3` to check the properties (precedence and type) for a few predefined operators of your own choice.
- c) Define a new binary operator `&` with precedence 950 (which is slightly less than the comma which has 1000) and `xfy` for the type. The operator should just make a call (using `call/1`) on two goals. Use again `current_op/3` to check the properties of the newly defined operator.

Example:

```
?- write(a) & write(b) & write(c).  
abc.
```

- d) Start with the following simple fact: `is_bigger_than(elephant,mouse)` and make `is_bigger_than` an operator with precedence 300 and `xfx` for the type. You should now be able to use infix notation rather than prefix notation:

```
?- elephant is_bigger_than mouse.  
true.
```

Make a guess, what is the expected outcome of

```
?- elephant is_bigger_than mouse = is_bigger_than(elephant,mouse).
```

Exercise 11.2 [Truth table of Boolean functions¹]

The goal of this exercise is to print the truth table of an arbitrary Boolean function as shown below.

```
?- tt(x or (not y and z)).
```

[x,y,z]	x or not y and z
[false,false,false]	false
[false,false,true]	true
[false,true,false]	false
[false,true,true]	false
[true,false,false]	true
[true,false,true]	true
[true,true,false]	true
[true,true,true]	true

For this, provide the required functionality by adding

- 1) definitions for the binary operators `and` and `or`, and for the unary operator `not` together with new predicates `bool_and/3`, `bool_or/3`, `bool_not/2` that explain the semantics of Boolean operations.
- 2) a new predicate `find_vars/2` that finds all Boolean variables present in a Boolean expression and puts them into a list.
- 3) a mechanism for enumerating over all possible evaluations of a given set of variables.
- 4) a new predicate `evaluate_expression/4` that allows evaluating a given Boolean expression: given an expression and two lists, one for the set of variables and one for the truth values of the variables, the predicate “returns” the truth value `true` or `false`.
- 5) a predicate `tt/1` that prints the truth table as shown above.

Exercise 11.3 [Call of an arbitrary function]

Write a predicate `map/3` that allows for applying an arbitrary function to all elements of a list. E.g., given the two predicates

```
neg(A,B)      :- B is -A.  
square(A,B)   :- B is A**2.
```

for negating and squaring one element, the `map/3` predicate should behave as indicated in the examples below:

```
?- map(neg,[-1,2,-3,-4,5],Result).  
Result = [1, -2, 3, 4, -5].  
  
?- map(square,[-1,2,-3,-4,5],Result).  
Result = [1, 4, 9, 16, 25].
```

Hint: A possible solution can be based on the `call/1` predicate and operator `=..`.

¹This exercise is inspired by the online Prolog tutorial of J.R. Fisher, available at https://www.cpp.edu/~jrffisher/www/prolog_tutorial/contents.html

Exercise 11.4 [Term analysis]

The goal of this exercise is to analyze a given term T and provide a user information on the type and properties of T .

a) provide a predicate `analyse_term/1` that prints information on the given term T and recognizes the following types and properties of T :

- T is an atom or ground term (cf. `atom/1` and `ground/1`)
- T is of type string (cf. `string/1`)
- T is compound or atomic (cf. `atomic/1` and `compound/1`)
- T is a variable/ not a variable (cf. `var/1` and `nonvar/1`)
- T is a number of a certain type (cf. `number/1`, `integer/1`, `float/1`, and `rational/3`)
- T is cyclic/ acyclic (cf. `cyclic/1` and `acyclic/1`)
- T is callable (cf. `callable/1`)

Examples:

```
?- analyse_term(a).  
found non-variable: a  
found an atom: a  
a is atomic.  
a is callable.  
a is a ground term.  
a is an acyclic term.
```

```
?- analyse_term(42).  
found non-variable: 42  
42 is a number.  
42 is an integer.  
42 is a rational  
42 is atomic.  
42 is a ground term.  
42 is an acyclic term.
```

```
?- analyse_term("42").  
found non-variable: 42  
found a string: 42  
42 is atomic.  
42 is a ground term.  
42 is an acyclic term.
```

```
?- analyse_term(X).  
found unbound variable: _G191  
_G191 is an acyclic term.
```

```
?- X=42, analyse_term(X).  
found non-variable: 42  
42 is a number.  
42 is an integer.  
42 is a rational.  
42 is atomic.  
42 is a ground term.  
42 is an acyclic term.  
X = 42.
```

```
?- analyse_term([a,b,c]).  
[a,b,c] is a compound term.  
[a,b,c] is callable.  
[a,b,c] is a ground term.  
[a,b,c] is an acyclic term.
```

```
?- analyse_term(3+X).  
3+_G192 is a compound term.  
3+_G192 is callable.  
3+_G192 is an acyclic term.
```

b) enrich the predicate from part a) with a recursive treatment for all operants of composed terms. Hints: you may use the Univ predicate `=.. /2` for creating a list containing the functor/operator and all its arguments. The predicates `compound_name_arguments/3`, `compound_name_arity/3`, `functor/3`, `arg/3` may also be very helpful.

Examples:

<pre> ?- analyse_term(3+X). 3+_G192 is a compound term. found functor: +/2 found non-variable: 3 3 is a number. 3 is an integer. 3 is a rational. 3 is atomic. 3 is a ground term. 3 is an acyclic term. found unbound variable: _G192 _G192 is an acyclic term. </pre>	<pre> ?- analyse_term([a]). [a] is a compound term. found functor: []/2 found non-variable: a found an atom: a a is atomic. a is callable. a is a ground term. a is an acyclic term. found non-variable: [] [] is atomic. [] is a ground term. [] is an acyclic term. </pre>
--	---

Exercise 11.5 [Arbitrary function evaluation]

Provide a predicate `func/1` that allows to compute the value of an arbitrary arithmetic function given as a parameter of the predicate. In case that the function contains free (i.e., unbound) variables, the user should be asked to provide a numeric constant value.

Examples:

```

?- func(4.2).
4.2 = 4.2

```

```

?- func(3*X).
found unbound variable: _G335
|: 12.
3*12 = 36
X=12.

```

```

?- func(X**2+2*Y-X).
found unbound variable: _G334
|: 2.
found unbound variable: _G338
|: 3.
2**2+2*3-2 = 8
X = 2, Y = 3.

```

Hint: For a possible solution the following predicates and operators are very helpful: `compound/1`, `compound_name_arguments/3`, `functor/3`, `=...`